	UNIVERSIDAD DON BOSCO **DEPARTAMENTO DE CIENCIAS BÁSICAS**
Ciclo III 2019	**Métodos Numéricos** **Guía de Laboratorio No. 0** **“Introducción a la programación en Matlab”**

I. RESULTADOS DE APRENDIZAJE

- Se familiariza con el entorno de trabajo de Matlab
- Conoce la sintaxis de las estructuras de control en Matlab
- Resuelve problemas de aplicación empleando los recursos de cálculo proporcionados por Matlab.

II. INTRODUCCIÓN TEORICA

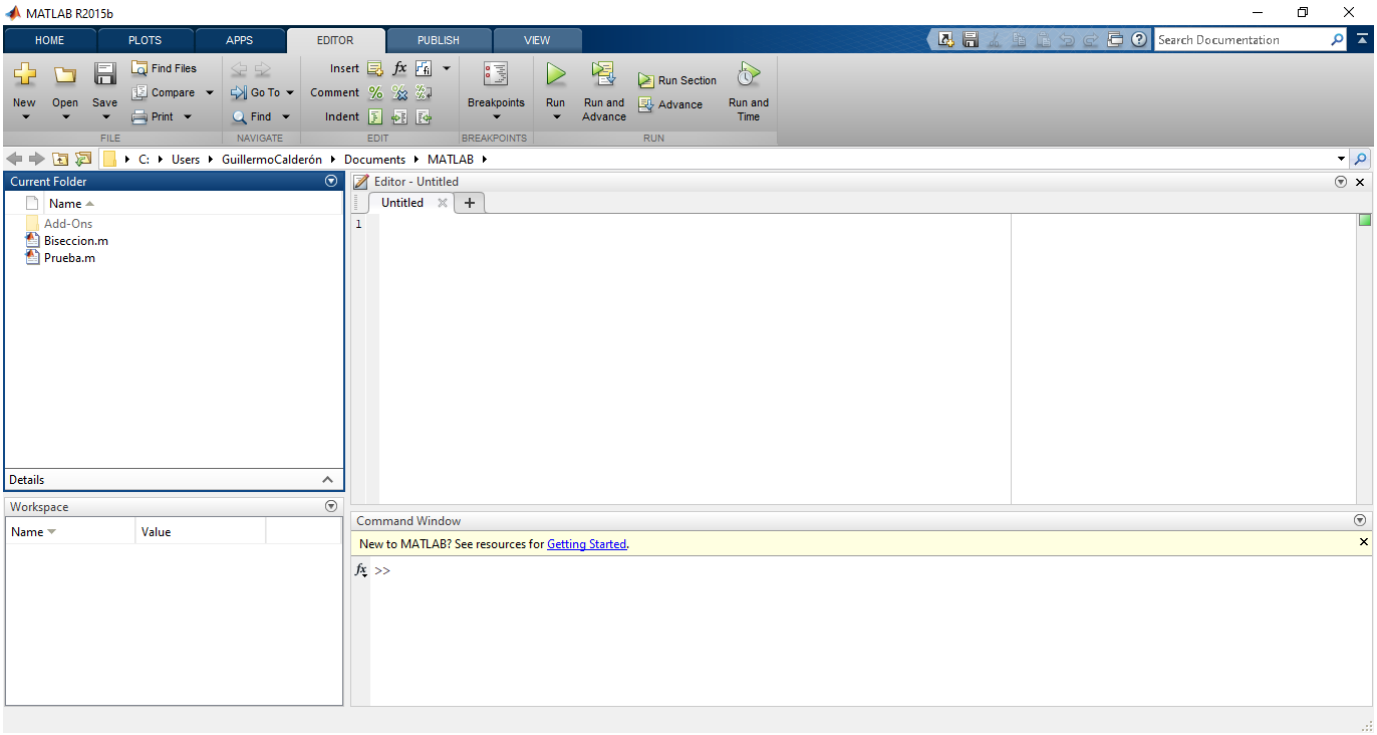
1. Introducción a MATLAB

Hoy en día, MATLAB es un herramienta de cálculo y lenguaje de programación matemático de alto nivel integrado con entorno gráfico amigable, visualización de datos, funciones, graficas 2D y 3D, procesamiento de imágenes, video, computación numérica para desarrollar algoritmos matemáticos con aplicaciones en ingeniería y ciencias exactas. Particularmente, en ingeniería es una herramienta muy poderosa para realizar aplicaciones en mecatrónica, robótica, control y automatización.

MATLAB es un acrónimo que proviene de **matrix laboratory** (laboratorio matricial) creado por el profesor y matemático Cleve Moler en 1970. Las versiones actuales de MATLAB se caracterizan por ser multiplataforma, es decir, se encuentra disponible para sistemas operativos Linux, Windows y Apple Mac OS. MATLAB posee varias características computacionales y visuales, entre las que sobresale la caja de herramientas (toolbox) la cual representa un amplio compendio de funciones y utilerías para analizar y desarrollar una amplia gama de aplicaciones en las áreas de ingeniería y ciencias exactas.

1.1 Componentes

El ambiente de programación de MATLAB es amigable al usuario, y está compuesto por una interfaz gráfica con varias herramientas distribuidas en ventanas que permiten programar, revisar, analizar, registrar datos, utilizar funciones, historial de comandos, etc.



1.2 Lenguaje

El lenguaje de programación de MATLAB es de alto nivel y permite programar matrices, arreglos e incorporar instrucciones de control de flujo del programa, funciones, comandos y estructuras de datos.

MATLAB contiene una diversidad de comandos que facilitan al usuario la implementación del problema a simular. Los comandos son funciones muy específicas que tienen un código depurado y que no se encuentra disponible al usuario.

1.3 Gráficos

MATLAB contiene un enorme número de funciones que facilitan la representación gráfica y visualización de variables, funciones, vectores, matrices y datos que pueden ser graficados en 2 y 3 dimensiones. Incluye también funciones de alto nivel para el análisis y procesamiento de gráficas, datos experimentales o de simulación, video e imágenes, animación y presentación de sólidos, así como el desarrollo de aplicaciones con interface

gráfica para incluir menús con botones, barras deslizadoras, diagramas a bloques, instrumentos de medición y ventanas para seleccionar opciones o herramientas de la aplicación.

1.4 Inicio

Una vez instalado MATLAB, la forma más simple de interaccionar con este paquete es introduciendo expresiones directamente en la ventana de comandos, por ejemplo iniciando con el tradicional “Hola mundo” sobre el *prompt*:

```
>> disp('Hola Mundo') ↲

Hola Mundo
>> 9+9 ↲

ans =

    18

>> sin(10) ↲

ans =

   -0.5440
```

Cuando no se especifique el nombre de la variable, MATLAB usará el nombre por defecto *ans* que corresponde a un nombre corto de `answer`. Al especificar una variable, el resultado se desplegará con el nombre de esa variable, por ejemplo:

```
>> z=9+9  ↲

z =

    18

>> y=sin(10) ↲

y =

   -0.5440
```

MATLAB funciona como calculadora, sobre el *prompt* de la ventana de comandos se pueden escribir expresiones aritméticas mediante los operadores “+”, “-“, “*”, “/”, “^” para la suma, resta, multiplicación y división.

Por ejemplo, similar a una calculadora se pueden evaluar expresiones como:

```
>> (10.3+8*5/3.33)^5 ↲
ans =

   5.5296e+06

>> ans^2  ↲

ans =

   3.0576e+13
```

MATLAB utiliza el operador **;** para deshabilitar la opción de desplegado en la ventana de comandos. Por ejemplo:

```
>> z=9+9; ↲
>>
```

es decir, con el operador ; MATLAB no exhibirá ningún valor de variable o función.

El lenguaje MATLAB permite insertar comentarios usando el operador **%** el cual deberá emplearse por cada línea de comentarios. Por ejemplo:

```
>> w= 10+20 % Suma de dos enteros

w =

    30

>> sign(-8999.4) % Obtiene el signo de un numero

ans =

    -1
```

Observe que al ejecutarse la instrucción o la sentencia el comentario no se despliega. Los comentarios sirven para documentar aspectos técnicos de un programa.

2. Lenguaje

El lenguaje de programación de MATLAB está compuesto por un conjunto de reglas gramaticales y sintaxis para escribir correctamente las variables, operadores, expresiones, funciones y todos los elementos que forman parte de la programación.

Las variables, constantes, operadores y funciones forma una expresión la cual será procesada por un analizador léxico y sintáctico antes de ser ejecutada por la computadora. A diferencia de otros lenguajes, las expresiones en MATLAB involucran matrices.

2.1 Variables

En el lenguaje de MATLAB las variables no requieren ningún tipo de declaración o definición. Esto tiene varias ventajas ya que facilita la programación. Cuando MATLAB encuentra el nombre de la nueva variable, automáticamente crea la variable y le asigna una localidad de memoria.

Los nombres usados para referenciar a las variables, funciones y otros tipos de objetos o estructuras definidos por el usuario se les conocen como **identificadores**. Los nombres de los identificadores constan de una letra como caracter inicial seguido de cualquier número de letras, dígitos y guiones bajos (_). MATLAB distingue las letras mayúsculas de las minúsculas; por ejemplo, la variable `A` es diferente a la variable `a`. La longitud de los nombres de las variables puede ser cualquier número de caracteres. Sin embargo, MATLAB usa los primeros 63 caracteres del nombre e ignora los restantes. Por lo tanto, para distinguir variables es importante escribir sus nombres con un máximo de 63 caracteres.

MATLAB puede trabajar con varios tipos de variables: en punto flotante, flotante doble, enteros, enteros largos, enteros cortos, tipo char, cadenas de caracteres, expresiones simbólicas, etc. Los diversos tipos de variables son interpretados como matrices.

Para saber qué tipo de variables estamos usando podemos utilizar el comando `whos` como en los ejemplos siguientes, primero se declaran las variables y posteriormente se una la función `whos.`

```
>> i=9; ↲
>> x=3.35; ↲
>> cadena='Guia 1'; ↲
>> A=[2 3; 3 5] ↲
```

```
A =

     2     3
     3     5

>> whos ↲
  Name        Size            Bytes  Class     Attributes

  A           2x2                32  double
  cadena      1x6                12  char
  i           1x1                 8  double
  x           1x1                 8  double
```

La columna `Size` indica la cantidad de memoria asignada a esa variable, mientras que la columna `Class` indica el tipo de variable. Observe que MATLAB trata a todas las variables como matrices para propósitos de programación.

Al terminar la sesión, o cuando ya no se usen las variables debido a que se desea trabajar con otro programa es recomendable limpiar todas las variables para liberar memoria. Para realizar lo anterior, se puede hacer con cada variable, por ejemplo `clear i,` posteriormente `clear x` y finalmente `clear cadena.` Otra forma de realizarlo es limpiando simultáneamente las tres variables a través de: `clear i x` cadena o cuando hay una gran cantidad de variables, entonces es conveniente usar `clear all` para limpiar todas las variables de forma simultánea.

2.2 Formato numérico

La forma de desplegar funciones, variables y cualquier tipo de dato por MATLAB se realiza a través del comando `format` el cual controla el formato numérico de los valores desplegados. Es decir, modifica el número de dígitos para el desplegado de los datos. Este comando solo afecta a los números que son desplegados, no al proceso de cálculo o al registro de las variables o datos.

```
>> format short
>> x=[9/8 8.3456e-8]
x =
    1.1250    0.0000
```

Observe que en el caso de la constante `8.3456e-8` lo despliega como 0.0000 debido a los límites del formato corto `format short.`

Ahora note la diferencia con `format short e`

```
>> format short e
>> x

x =

   1.1250e+00   8.3456e-08
```

Para desplegar la información de x con mayor número de dígitos podemos utilizar el `format long`

```
>> format long
>> x
x =
   1.125000000000000   0.000000083456000
```

El formato numérico empleado por defecto en MATLAB es el formato corto (short): `format short`. Sin embargo, es posible utilizar otros tipos de formatos como: extendido, notación científica, punto fijo, punto flotante, formato de ingeniería, etc.

Los formatos numéricos disponibles para el comando `format` se encuentran resumidos en la siguiente tabla:

Tipo	Resultado	Ejemplo: >> pi
format short	Formato coma fija con 4 dígitos después de la coma (es el formato que viene por defecto)	3.1416
format long	Formato coma fija con 14 o 15 dígitos después de la coma	3.14159265358979
format short e	Formato coma flotante con 4 dígitos después de la coma	3.1416e+000
format long e	Formato coma flotante con 14 o 15 dígitos después de la coma	3.141592653589793e+000
format short g	La mejor entre coma fija o flotante con 4 dígitos después de la coma	3.1416
format long g	La mejor entre coma fija o flotante con 14 o 15 dígitos después de la coma	3.14159265358979
format short eng	Notación científica con 4 dígitos después de la coma y un exponente de 3	3.1416e+000
format long eng	Notación científica con 16 dígitos significantes y un exponente de 3	3.14159265358979e+000
format bank	Formato coma fija con 2 dígitos después de la coma	3.14
format hex	Hexadecimal	400921fb54442d18
format rat	Aproximación racional	355/113
format +	Positivo, negativo o espacio en blanco	+

2.3 Operadores

Los operadores en MATLAB juegan un papel determinante ya que manipulan a las variables y funciones. Adicional a los operadores aritméticos básicos hay operadores específicos para funciones, operaciones lógicas, relacionales, estructuras de datos, uniones y matrices.

A continuación se presentan diversos tipos de operadores de utilidad en MATLAB.

- **Operador colon :**

El operador colon: (dos puntos) es uno de los operadores más importantes para programar, se emplea para diferentes propósitos, como por ejemplo en MATLAB técnicamente se conoce como vectorización al proceso de generar una secuencia de números usando `1:10`.

```
>> 1:10

ans =

     1     2     3     4     5     6     7     8     9    10
```

En este caso el incremento es de uno en uno. Si se requiere un paso de incremento especifico, por ejemplo 2, entonces se procede de la siguiente forma:

```
>> 1:2:10
ans =
     1     3     5     7     9
```

El valor del paso de incremento también puede ser menor que 1, por ejemplo:

```
>> 1:0.1:2
ans =
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000
1.7000    1.8000    1.9000    2.0000
```

El valor de paso de incremento también puede ser negativo

```
>> 10:-1:2
ans =
    10     9     8     7     6     5     4     3     2
```

- **Operador semicolon ;**

El operador semicolon o punto y coma ; tiene varias funciones. Una de ellas se encuentra relacionada con desplegar el resultado que tienen las variables, constantes, funciones o gráficas. Cuando se inserta al final de la expresión, instrucción o comando se inhabilita el desplegado. Por ejemplo:

```
>> w=sin(pi/2);

>>
```

Si el operador ; no se coloca al final de la instrucción, entonces se produce el desplegado del valor de la variable w.

```
>> w=sin(pi/2)

w =

     1
```

Otra funcionalidad del operador ; es generar filas en matrices. Por ejemplo:

```
>> A=[19 3; 4 5]
A =
    19     3
     4     5
```

El operador ; que precede al número 3 y antecede al 4 genera una fila de esta matriz.

- **Operador ,**

El operador coma , tiene más de una función en MATLAB. Por ejemplo, cuando se emplea en funciones indica la separación de los argumentos como en el caso de `y=sin(t,x).` Para referenciar los elementos de una matriz se especifica el número de fila y de columna separados por una coma `A(3,4).`

- **Operador ‘**

El operador ‘ se relaciona con el manejo de datos tipo char o cadena de caracteres. También representa la transpuesta de una matriz.

```
>> cadena='Una cadena de letras'
cadena =
Una cadena de letras
```

En matrices, el operador ‘ representa la matriz transpuesta, es decir A’ = A^T

```
>> M=[3 5 7; 1 8 9; 3 2 9]

M =

     3     5     7
     1     8     9
     3     2     9

>> M'

ans =

     3     1     3
     5     8     2
     7     9     9
```

- Operador ~

El operador tilde ~ se emplea para deshabilitar una variable de salida de una función. Es muy útil cuando la función retorna más de una variable y no se requiere usar todas las variables; supóngase que la función `control_robot` retorna dos variables (`error` y `par`) y la sintaxis es: `[error, par]= control_robot(q).` No se requiere usar la variable `error`, únicamente `par`, entonces se usa de la siguiente forma:

```
[~ , par] = control_robot(q)
```

También se emplea como negación en operadores lógicos, por ejemplo en ~ = que significa **no es igual a.**

3. Matrices

En MATLAB todas las variables son matrices. Las matrices se introducen de una manera directa:

```
>> A= [ 1 2 3; 4 5 6; 7 8 9]


A =
     1     2     3
     4     5     6
     7     8     9
```

Los puntos y comas separan las filas de la matriz, mientras que los elementos de la misma fila deben separarse mediante un espacio en blanco o una coma.

Los elementos de una matriz se denotan en MATLAB por `A(i, j)` donde `i` representa el número de fila y `j` el número de columna del elemento. Por ejemplo, `A(2, 3)` hace referencia al elemento de la matriz A que se ubica en la segunda fila y tercera columna.

```
>> A(2,3)
ans =
     6
```

Es muy común cometer errores al referenciar los elementos o entradas de una matriz. Por ejemplo, los elementos A(i, j) de una matriz de orden $n \times p$ deben satisfacer lo siguiente:

- No usar en la referencia de los elementos de una matriz A la posición $A(0,0)$ o números negativos $A(-3,-5)$, ni números reales como $A(1.32, 2.67)$. En todos estos ejemplos existe error de sintaxis y la ejecución de un programa no puede realizarse hasta no corregir dicho error.
- Los pivotes i, j son números acotados por $0 < i \leq n$, $0 < j \leq p$. Por ejemplo si A es de orden 3×5, existe un error en la referencia del elemento $A(4, 6)$.
- Los pivotes i, j son números enteros positivos (números naturales).

Generando matrices básicas

La siguiente tabla muestra las opciones más comunes que permiten generar matrices básicas:

Nombre de la función	Descripción y características
`zeros (n)`	Crea una matriz cuadrada n x n de ceros.
`zeros (m,n)`	Crea una matriz m x n de ceros.
`ones (n)`	Crea una matriz cuadrada n x n de unos.
`ones (m,n)`	Crea una matriz m x n de unos.
`rand (n)`	Crea una matriz cuadrada n x n de números aleatorios con distribución uniforme (0,1).
`rand (m,n)`	Crea una matriz m x n de números aleatorios con distribución uniforme (0,1).
`randn (n)`	Crea una matriz cuadrada n x n de números aleatorios con distribución normal (0,1).
`randn (m,n)`	Crea una matriz m x n de números aleatorios con distribución normal (0,1)
`eye (n)`	Crea una matriz cuadrada n x n de unos en la diagonal y ceros el resto
`eye (m,n)`	crea una matriz m x n de unos en la diagonal y ceros el resto

Ejemplos

```
>> A=zeros(3,4)

A =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
>> B=10*ones(4,3)
B =
    10    10    10
    10    10    10
    10    10    10
    10    10    10

>> C=randn(3)
C =
    0.5377    0.8622   -0.4336
    1.8339    0.3188    0.3426
   -2.2588   -1.3077    3.5784
```

Operaciones básicas con matrices

Símbolo	Expresión	Operación
+	A + B	Suma de matrices
-	A - B	Resta de matrices
*	A * B	Multiplicación de matrices
.*	A .* B	Multiplicación elemento a elemento de matrices
/	A / B	División de matrices
./	A ./ B	División elemento a elemento de matrices
^	A ^ n	Potenciación (n debe ser un número, no una matriz)
.^	A .^ B	Potenciación elemento a elemento de matrices
'	A '	Trasposición compleja conjugada
.'	A .'	Trasposición de matrices

Ejemplo: Definamos 3 matrices para trabajar con ellas.

```
>> A=[1 2 ; 3 4]
A =
     1     2
     3     4

>> B=[ 1 1; 0 1]
B =
     1     1
     0     1
```

```
>> C=[1+1i 2+2i ; 3+1i 4+7i]
C =
   1.0000 + 1.0000i   2.0000 + 2.0000i
   3.0000 + 1.0000i   4.0000 + 7.0000i

>> A+B % Suma de matrices
ans =
     2     3
     3     5

>> A-B %Resta de matrices
ans =
     0     1
     3     3

>> A*B %Multiplicacion de matrices
ans =
     1     3
     3     7

>> A.*B % Multiplicacion elemento por elemento
ans =
     1     2
     0     4

>> C' % Transpuesta conjugada
ans =
   1.0000 - 1.0000i   3.0000 - 1.0000i
   2.0000 - 2.0000i   4.0000 - 7.0000i

>> C.' %Transpuesta
ans =
   1.0000 + 1.0000i   3.0000 + 1.0000i
   2.0000 + 2.0000i   4.0000 + 7.0000i
```

```
>> A + 2 %Si sumamos 2 a la matriz se suma elemento por elemento
ans =
     3     4
     5      6
```

4. Funciones matemáticas comunes

Aproximaciones

Función	¿Qué hace?	Ejemplo x = 5.92
ceil (x)	redondea hacia infinito	6
fix (x)	redondea hacia cero	5
floor (x)	redondea hacia menos infinito	5
round (x)	redondea hacia el entero más próximo	6

(con **x** escalar, vector o matriz, pero redondearía en cada caso los elemento individualmente)

Ejemplos:

```
>> round([19.5464 13.654 -2.1565 0.75 14.423])
ans =
    20    14    -2     1    14
```

Trigonometría

Función	¿Qué hace?
… (x)	función trigonométrica con el ángulo expresado en radianes
sin (x) **cos (x)** **tan (x)** **csc (x)** **sec (x)** **cot (x)**	seno (radianes) coseno tangente cosecante secante cotangente
…d (x)	función trigonométrica con el ángulo expresado en grados
sind (x) …	seno (grados) …
…h (x)	función trigonométrica hiperbólica con el ángulo expresado en radianes
sinh (x) …	seno hiperbólico (radianes) …

Función	¿Qué hace?
a… (x)	inversa de la función trigonométrica con el resultado expresado en radianes
asin (x) …	arco seno (radianes) …
a…d (x)	inversa de la función trigonométrica con el resultado expresado en grados
asind (x) …	arco seno (grados) …
a…h (x)	inversa de la función trigonométrica hiperbólica con el resultado expresado en radianes
asinh (x) …	arco seno hiperbólico (radianes) …

Ejemplos:

```
>> sin(pi/2)
ans =
     1

>> sind(-90)
ans =
    -1

>> cosd(60)
ans =
    0.5000

>> asind(1)
ans =
    90
```

Algunas operaciones

Función	¿Qué hace?
abs (x)	valor absoluto o magnitud de un número complejo
sign (x)	signo del argumento si x es un valor real (-1 si es negativo, 0 si es cero, 1 si es positivo)
exp (x)	exponencial
gcd (m,n)	máximo común divisor
lcm (m,n)	mínimo común múltiplo
log (x)	logaritmo neperiano o natural
log2 (x)	logaritmo en base 2
log10 (x)	logaritmo decimal
mod(x,y)	módulo después de la división
rem (x,y)	resto de la división entera
sqrt (x)	raíz cuadrada
nthroot (x,n)	raíz n-ésima de x

(**x** e **y** cualquier escalar, **m** y **n** enteros)

```
>> abs(-7) % Valor absoluto de -7
ans =
     7

>> sign(10)  %Signo del numero 10
ans =
     1

>> gcd(9,12)  % Maximo común divisor
ans =
     3

>> lcm(10,25) % Minimo comun multiplo
ans =
    50
```

```
>> mod(-12,5) % Modulo de la division de -12 entre 5
ans =
     3

>> rem(12,5) % resto de la division 12 entre 5
ans =
     2

>> nthroot(8,3) %Raiz cubica de 8
ans =
     2
```

5. Operadores relacionales y lógicos

Como entradas a las expresiones relacionales y lógicas, Matlab considera que cero es falso y que cualquier número distinto de cero es verdadero. La salida de expresiones de este tipo produce 1 si es verdadero y 0 si es falso.

Operadores relacionales

Operador	¿Qué significa?
<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual a
~=	distinto de

La salida de las operaciones lógicas se puede utilizar también en operaciones matemáticas.

Operadores lógicos

Operador	¿Qué significa?
&	y
\| | o |
~ | no |

6. Programación en MATLAB

Un programa es una secuencia de instrucciones, expresiones, funciones, comandos y declaraciones para realizar aplicaciones. En MATLAB las instrucciones son ejecutadas una tras otra, en forma secuencial, es decir, en el mismo orden en el que van apareciendo. La secuencia del programa depende de las instrucciones que controlan el flujo del programa, dependiendo de ciertas condiciones se ejecuta un determinado conjunto de instrucciones o bloque del programa.

El lenguaje de programación MATLAB es muy rico en instrucciones y funciones, ya que cuenta con un amplio compendio de librerías para diversas áreas de ingeniería y ciencias exactas.

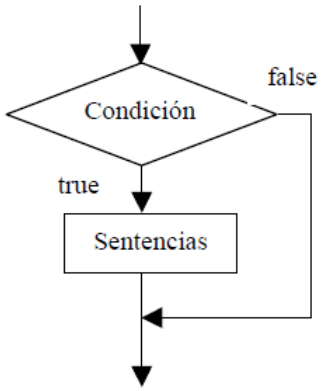
6.1 Estructuras condicionales

Una instrucción condicional permite a MATLAB realizar decisiones para ejecutar un grupo de funciones, comandos, etc. Si la condición en la estructura condicional es verdadera, entonces se realiza un grupo de expresiones del programa. Si la condición es falsa, entonces no ejecuta o salta ese grupo de expresiones.

Estructura if

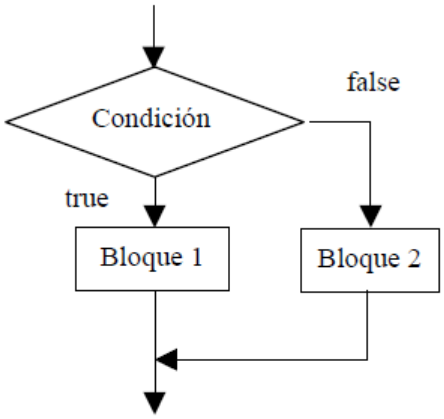
En su forma más simple, la sentencia if se escribe en la forma siguiente (obsérvese que a diferencia de C/C++/Java, la condición no va entre paréntesis, aunque se pueden poner si se desea).

```
if <condición>
    sentencias;
end
```



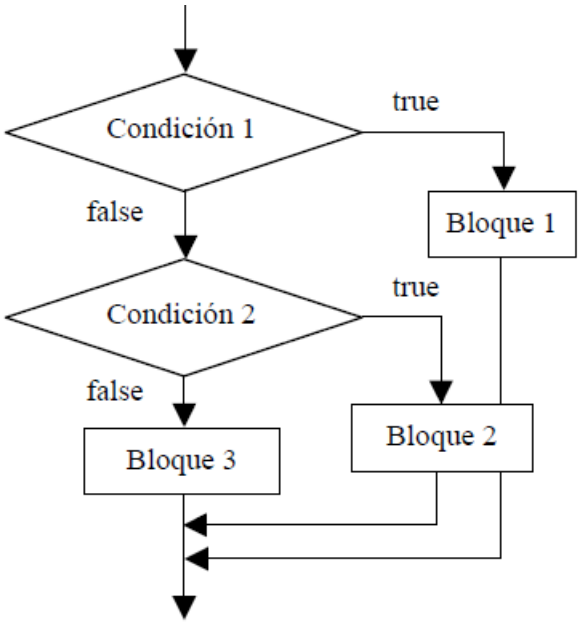
Si se quiere ejecutar un conjunto de instrucciones si la condición es verdadera y otro conjunto de instrucciones si la condición es falsa, se debe usar la estructura `if-else`, cuya sintaxis se muestra a continuación:

```
if <condición>
    sentencias1
else
    sentencias2
end
```



Existe también la *bifurcación múltiple*, en la que pueden concatenarse tantas condiciones como se desee, y que tiene la forma:

```
if <condicion1>
        sentencias1
elseif <condicion2>
        sentencias2
elseif <condicion3>
        sentencias3
else
      sentenciasN
end
```



Estructura switch

La estructura `switch` ejecuta un grupo de sentencias con base al valor de la variable o expresión que emplea al lado derecho de su sintaxis. Para dividir los grupos de sentencias se emplean las palabras claves `case` y `otherwise`. La instrucción `switch` finaliza con `end`.

```
switch selector
   case valor1
      sentencias1
   case valor2
      sentencias2
      ..........
   otherwise
      sentenciasN
end
```

MATLAB distingue los diferentes grupos de instrucciones asociados a un respectivo `case`. Por ejemplo, si entra a ejecutar el grupo de instrucciones del `case` 1, se sale de la instrucción `switch` cuando encuentra el `case` 2 y continúa con otro grupo de instrucciones ya fuera del `switch`.

6.2 Estructuras repetitivas

Estructura for

La sentencia ***for*** repite un conjunto de sentencias un número predeterminado de veces. La sentencia ***for*** de MATLAB es muy diferente y no tiene la generalidad de la sentencia ***for*** de C/C++/Java. La siguiente construcción ejecuta *sentencias* con valores de **i** de **1** a **n**, variando de uno en uno.

```
for i=1:n
    sentencias
end
```

En el siguiente ejemplo se presenta el caso más general para la variable del bucle (*valor_inicial: incremento: valor_final*); el bucle se ejecuta por primera vez con **i=n**, y luego **i** se va reduciendo de 0.2 en 0.2 hasta que llega a ser menor que 1, en cuyo caso el bucle se termina:

```
for i=n:-0.2:1
    sentencias
end
```

En el siguiente ejemplo se presenta una estructura correspondiente a dos ***bucles anidados***. La variable **j** es la que varía más rápidamente (por cada valor de **i**, **j** toma todos sus posibles valores):

```
for i=1:m
    for j=1:n
        sentencias
    end
end
```

Estructura while

La estructura del bucle ***while*** es muy similar a la de C/C++/Java. Su sintaxis es la siguiente:

```
while condicion
    sentencias
end
```

donde ***condicion*** puede ser una expresión vectorial o matricial. Las *sentencias* se siguen ejecutando mientras la condición sea verdadera. En MATLAB una condición se considera verdadera mientras haya elementos distintos de cero en ***condicion***, es decir, mientras haya algún o algunos elementos ***true***. El bucle se termina cuando *todos los elementos* de ***condicion*** son ***false*** (es decir, cero).

Sentencia break

Al igual que en C/C++/Java, la sentencia ***break*** hace que se termine la ejecución del bucle ***for*** y/o ***while*** más interno de los que comprenden a dicha sentencia.

Sentencia continue

La sentencia ***continue*** hace que se pase inmediatamente a la siguiente iteración del bucle ***for*** o ***while***, saltando todas las sentencias que hay entre el ***continue*** y el fin del bucle en la iteración actual.

6.3 Lectura y escritura de variables

Se verá a continuación una forma sencilla de leer variables desde teclado y escribir mensajes en la pantalla del PC.

Función input

La función ***input*** permite imprimir un mensaje en la línea de comandos de MATLAB y recuperar como valor de retorno un valor numérico o el resultado de una expresión tecleada por el usuario. Después de imprimir el mensaje, el programa espera que el usuario teclee el valor numérico o la expresión. Cualquier expresión válida de MATLAB es aceptada por este comando. El usuario puede teclear simplemente un vector o una matriz. En cualquier caso, la expresión introducida es evaluada con los valores actuales de las variables de MATLAB y el resultado se devuelve como valor de retorno.

```
>> n= input('Ingresa el valor de n')
```

Otra posible forma de esta función es la siguiente (obsérvese el parámetro 's'):

```
>> nombre= input('Ingresa tu nombre', 's')
```

En este caso el texto tecleado como respuesta se lee y se devuelve sin evaluar, con lo que se almacena en la cadena ***nombre***. Así pues, en este caso, si se teclea una fórmula, se almacena como texto sin evaluarse.

Función display

La función ***disp*** permite imprimir en pantalla un mensaje de texto o el valor de una matriz, pero sin imprimir su nombre. En realidad, ***disp*** siempre imprime vectores y/o matrices: las cadenas de caracteres son un caso particular de vectores. Considérense los siguientes ejemplos de cómo se utiliza:

```
display('El programa ha iniciado')
A= rand(4,4);
display(A)
```

Función frprintf

Esta función permite imprimir en pantalla una salida formateada. Su sintaxis es la siguiente:

```
fprintf('cadena con formato', expr1, expr2, ...)
```

El primer parámetro es una cadena de caracteres que se mostrará en la pantalla. La cadena puede contener especificadores de conversión que se sustituirán por el resultado de ir ´ evaluando las expresiones que siguen al primer parámetro. Los especificadores de conversión más utilizados son:

- %f para números con decimales.
- %d para enteros.
- %s para cadenas de caracteres.
- %c para un carácter.

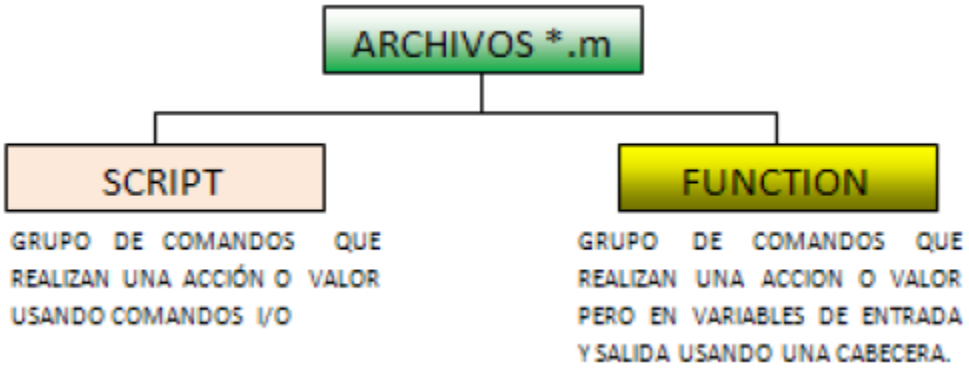
```
v = input('Introduce un vector con dos factores: ');
fprintf('%fx%f = %f\n', v(1), v(2), v(1)*v(2));
```

En este último ejemplo se ha incluido al final de la cadena de formato el car ´ acter de escape ´ \n, que representa un salto de línea. Los caracteres de escape más importantes son:

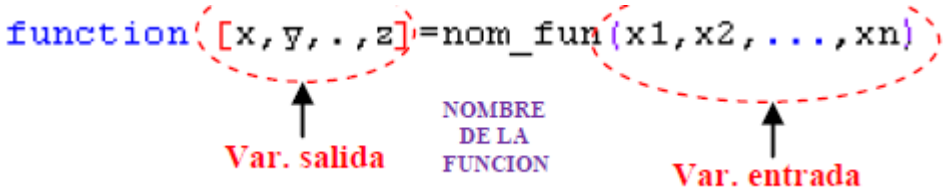
- \n salto de línea.
- \t tabulador horizontal.
- '' comilla simple.
- %% signo de porcentaje.
- \\ barra invertida.

6.4 Archivos .m

Estos son de dos clases: Script y Function



Cabeceras de una función



[Variables de salida] : si es más de un variable se separan por comas.
si es solo una se puede omitir los [ ]
(Variables de entrada) : si es más de una separadas por comas.
Las funciones (functions) se deben grabar como archivo m (M-file) y el nombre del archivo debe ser igual al nombre de la función.

III. PROCEDIMIENTO

Resolver los siguientes problemas usando archivos .m:

1. Realizar un programa que lea una temperatura en Celsius y determine su equivalente en Fahreinheit y Kelvin.

```
display('CONVERSOR DE TEMPERATURA');
celsius= input('Ingresa la temperatura en Celsius: ');
kelv= celsius+ 273.15;
fahr=(9/5)*celsius+32;
fprintf('La temperatura en Kelvin es: %f \n',kelv);
fprintf('La temperatura en  Fahreinheit es: %f \n', fahr);
```

Ejecución

```
CONVERSOR DE TEMPERATURA
Ingresa la temperatura en Celsius: 100
La temperatura en Kelvin es: 373.150000
La temperatura en  Fahreinheit es: 212.000000
```

2. Desarrollar un programa que lea un número natural n y determine su factorial.

```
n= input('Ingrese el valor de n: ');
prod=1; % Variable para acumular el producto
for i=1:n % Iterando de 1 a n
    prod=prod*i; % Acumulando los productos
end
fprintf('%d != %d \n', n, prod);
```

Ejecución

```
Ingrese el valor de n: 6
6 != 720
```

3. Elaborar un programa que lea un número natural n desde teclado y determine si este es primo.

```
n= input('Ingrese el valor de n: ');
div=0; % Contador de divisores
for i=1:n % Iterando de 1 a n
    if rem(n,i)==0 % Si el residuo de la division es 0
        div=div+1; % Aumentando el contador de divisores
    end
end
if div== 2 % Si n tiene 2 divisores
    fprintf('El número %d es primo \n', n);
else
    fprintf('El número %d NO es primo \n', n);
end
```

Ejecución

```
Ingrese el valor de n: 43
El número 43 es primo
```

4. Crear una función que permita obtener la suma de los primeros n términos de la serie de Taylor para aproximar el exponencial de un número real x dado el valor de n y el valor de x.

La serie de Taylor para la función exponencial es:

$$S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

```
% Parametro de salida: s (suma de terminos)
% Parametros de entrada: n (cantidad de terminos) y x
% Nombre de la función: taylor_exponencial
function s = taylor_exponencial( n, x )
s=1;
for i=1:n % Iterando de 1 a n
    s=s+(x^i)/factorial(i); % Acumulando la suma
end
end
```

Ejecución

```
>> taylor_exponencial(40,1)

ans =

   2.718281828459046
```

V. EJERCICIOS COMPLEMENTARIOS

1. Elaborar un programa que lea el valor de un número natural n y muestre la lista de los primeros n primos. Así, por ejemplo, si n= 6, el programa debería mostrar la siguiente lista de números primos: 2, 3, 5, 7, 11, 13.
2. Elaborar un programa que lea una matriz de números (de cualquier dimensión) y muestre los siguiente elementos:
    - Suma de los elementos de cada una de sus filas
    - Suma de los elementos de cada una de sus columnas
    - Adicionalmente, si la matriz es cuadrada, deberá mostrarse la suma de elementos de la diagonal principal.

Asi, por ejemplo, si la matriz ingresada es:

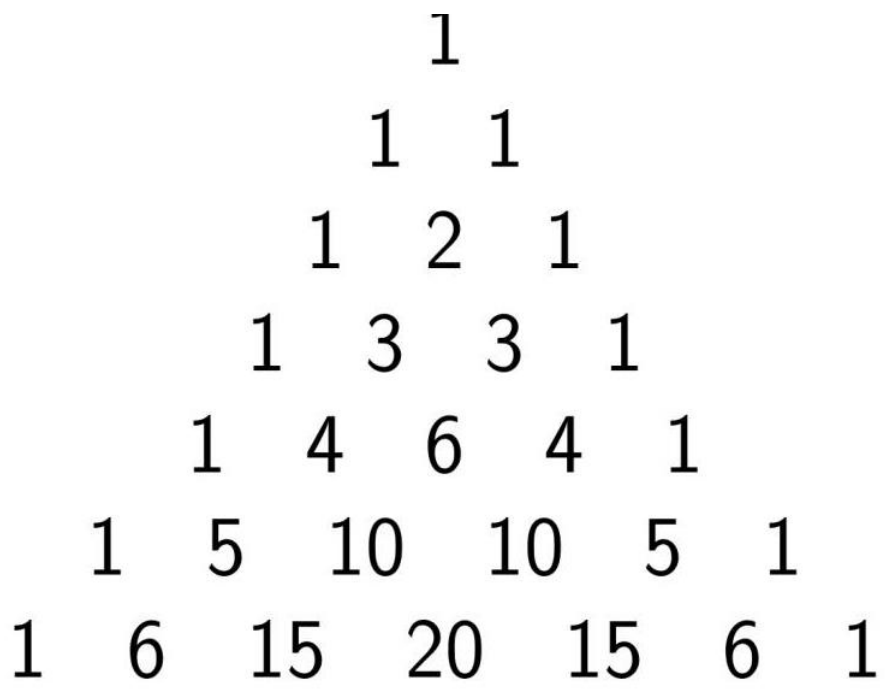
$$\begin{bmatrix} 2 & 3 & -5 \\ 7 & 0 & 9 \\ -2 & -5 & 5 \end{bmatrix}$$

El programa debería mostrar:

- Suma de elementos de la fila 1: 0
- Suma de elementos de la fila 2: 16
- Suma de elementos de la fila 3: -2
- Suma de elementos de la columna 1: 7
- Suma de elementos de la columna 2: -2
- Suma de elementos de la columna 3: 9
- Suma de elementos de la diagonal principal: 7

3. Elaborar un programa que lea el valor de un número natural n y muestre la lista de los primeros n elementos de la Serie de Fibonacci. Así, por ejemplo, si n= 10, el programa debería mostrar la siguiente serie: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

4. Elaborar un programa que lea el valor de un número natural n y muestre los primeros n niveles del triángulo de Pascal. Así, si n=7, el programa debería mostrar:



5. Crear una función que permita obtener la suma de los primeros n términos de la serie de Taylor para aproximar el coseno de un número x dado el valor de x y la cantidad de términos n.

La serie de Taylor para la función coseno es:

$$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$